

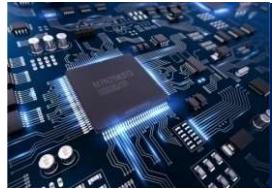
Cours Circuits Numériques

Fatiha El Hatmi

ISAMM

Cycle ingénieur en sciences appliquées et en technologie

2024/2025



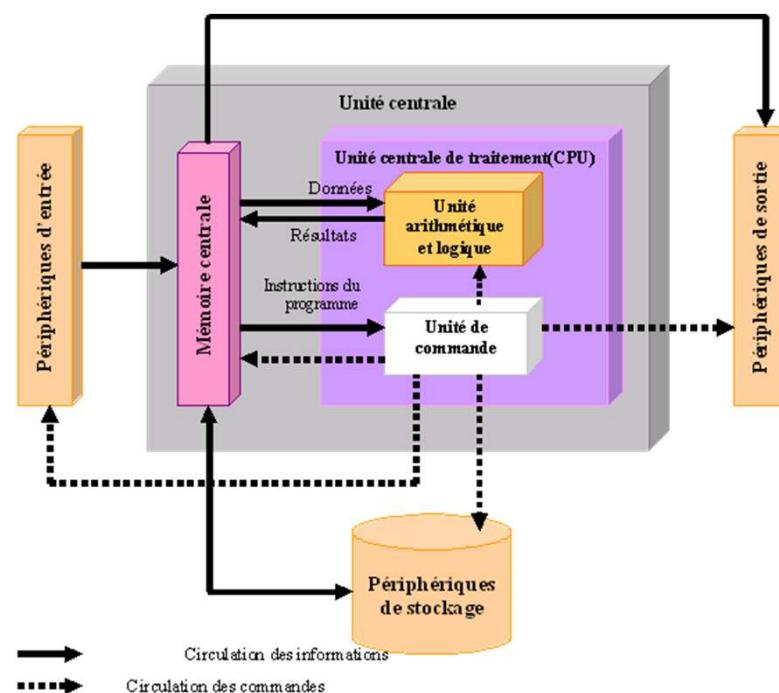
Plan du cours

- Chapitre 1: Historique des calculateurs
- Chapitre 2: Le transistor et portes logiques
- Chapitre 3: Représentation (codage) des données
- Chapitre 4: Circuits combinatoires
- Chapitre 5: Architecture de base d'un ordinateur
- Chapitre 6: Les mémoires
- Chapitre 7: Jeu d'instructions**



Chapitre VII

Jeu d'instructions





Plan chapitre VII

- 1. Introduction**
- 2. Type d'instructions**
- 3. Codage des instructions**
- 4. Registres sur 8086**
- 5. Les indicateurs (Flags)**
- 6. Gestion de la mémoire par le 8086**
- 7. Mode d'adressage**
- 8. Temps d'exécution des instructions**
- 9. Programmation assembleur**



Introduction

- Un programme est un ensemble **d'instructions exécutées** dans un **ordre** bien déterminé.
- Un programme est exécuté par un processeur (machine).
- Un programme est généralement écrit dans un langage évolué (Pascal, C, VB, Java, etc.).
- Les instructions qui constituent un programme peuvent être classifiées en 4 catégories :
 - Les Instructions **d'affectations** : permet de faire le transfert des données
 - Les instructions **arithmétiques et logiques**.
 - Les Instructions de **branchement** (conditionnelle et inconditionnelle)
 - Les Instructions **d'entrées sorties**.



Introduction

- Pour exécuter un programme par une machine, on passe par les étapes suivantes :
 1. Édition : on utilise généralement un **éditeur de texte** pour écrire un programme et le sauvegarder dans un fichier.
 2. Compilation : un compilateur est un programme qui convertit le **code source** (programme écrit dans un langage donné) en un programme écrit dans un **langage machine** (binaire). Une instruction en langage évolué peut être traduite en plusieurs instructions machine.
 3. Chargement : charger le programme en **langage machine** dans **mémoire** afin de l'exécuter .
- Comment **s'exécute** un programme dans la machine ?
- Pour comprendre le mécanisme d'exécution d'un programme → il faut comprendre le mécanisme de l'exécution d'une **instruction** .



Type d'instructions

- Chaque microprocesseur possède un **certain nombre limité** d'instructions qu'il peut exécuter. Ces instructions s'appellent **jeu d'instructions**.
- Le jeu d'instructions décrit l'ensemble des opérations élémentaires que le microprocesseur peut exécuter.
- Les instructions peuvent être classifiées en 4 catégories :
 - Instruction d'affectation : elle permet de faire le transfert des données entre les registres et la mémoire
 - Écriture : registre → mémoire
 - Lecture : mémoire → registre
 - Les instructions arithmétiques et logiques (ET , OU , ADD,.....)
 - Instructions de branchement (conditionnelle et inconditionnelle)
 - Instructions d'entrées sorties.

Couches ordinateur

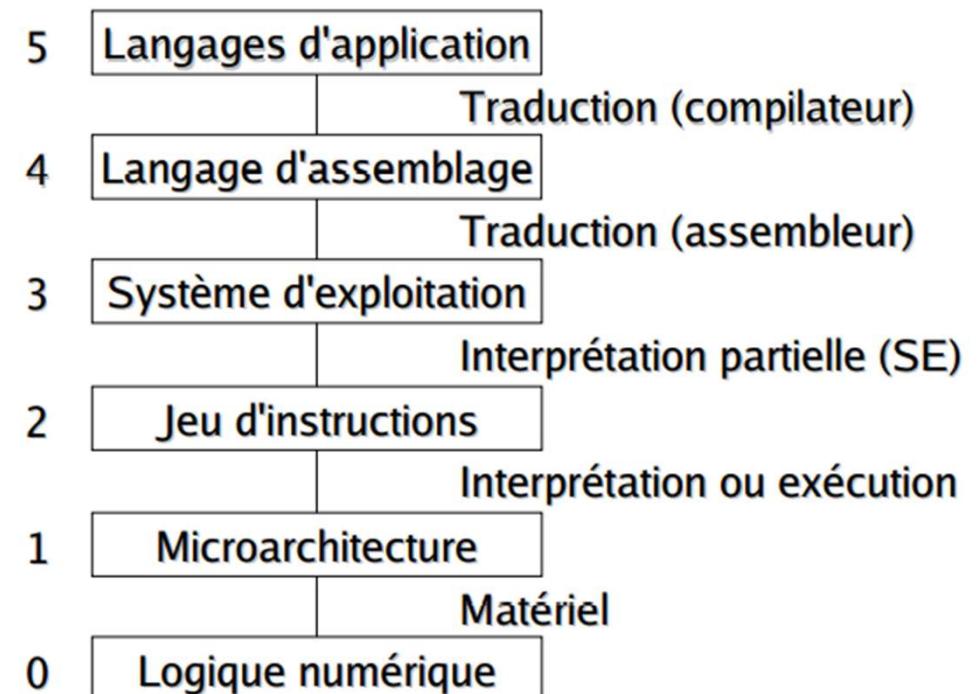
- Les ordinateurs modernes sont conçus comme un ensemble de couches
- Chaque couche représente une abstraction différente, capable d'effectuer des opérations et de manipuler des objets spécifiques
- L'ensemble des types de données, des opérations, et des fonctionnalités de chaque couche est appelée son « architecture »
- L'étude de la conception de ces parties est appelée « architecture des ordinateurs »

□ Couche logique numérique

- Les objets considérés à ce niveau sont les portes logiques, chacune construite à partir de quelques transistors
- Chaque porte prend en entrée des signaux numériques (0 ou 1) et calcule en sortie une fonction logique simple (ET, OU, NON)
- De petits assemblages de portes peuvent servir à réaliser des fonctions logiques telles que mémoire, additionneur, ainsi que la logique de contrôle de l'ordinateur

□ Couche microarchitecture

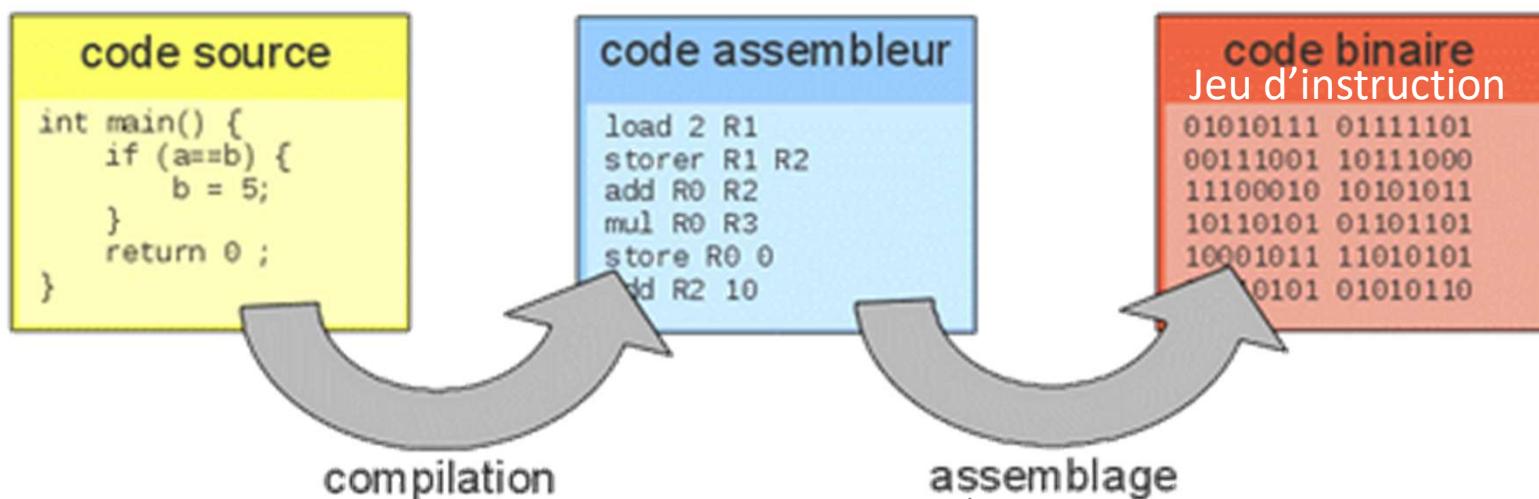
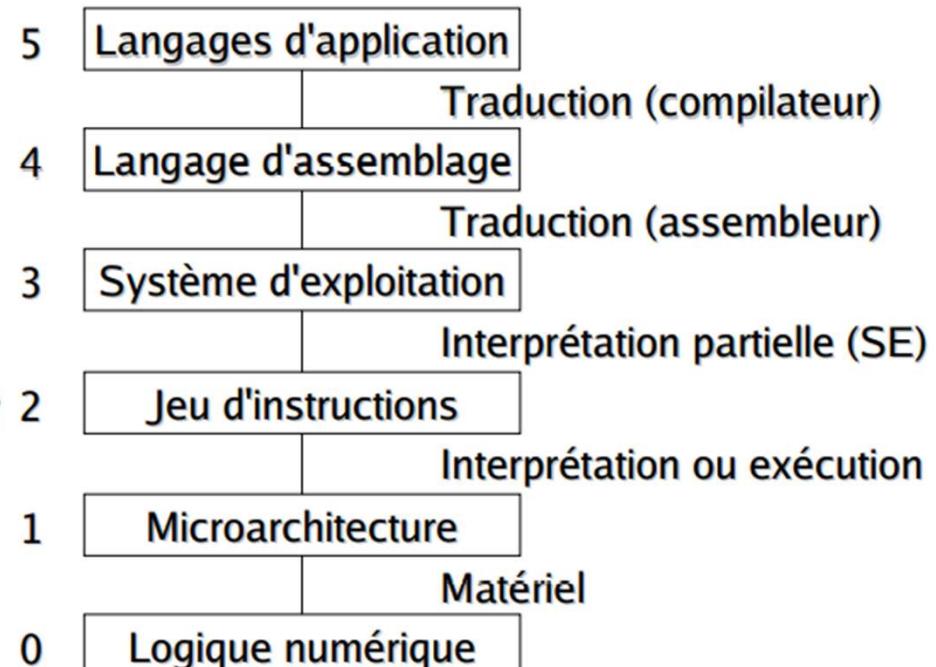
- On dispose à ce niveau de plusieurs registres mémoire et d'un circuit appelé UAL (Unité Arithmétique et Logique, ALU) capable de réaliser des opérations arithmétiques élémentaires
- Les registres sont reliés à l'UAL par un chemin de données permettant d'effectuer des opérations arithmétiques entre registres
- Le contrôle du chemin de données est soit microprogrammé, soit matériel



Couches ordinateur

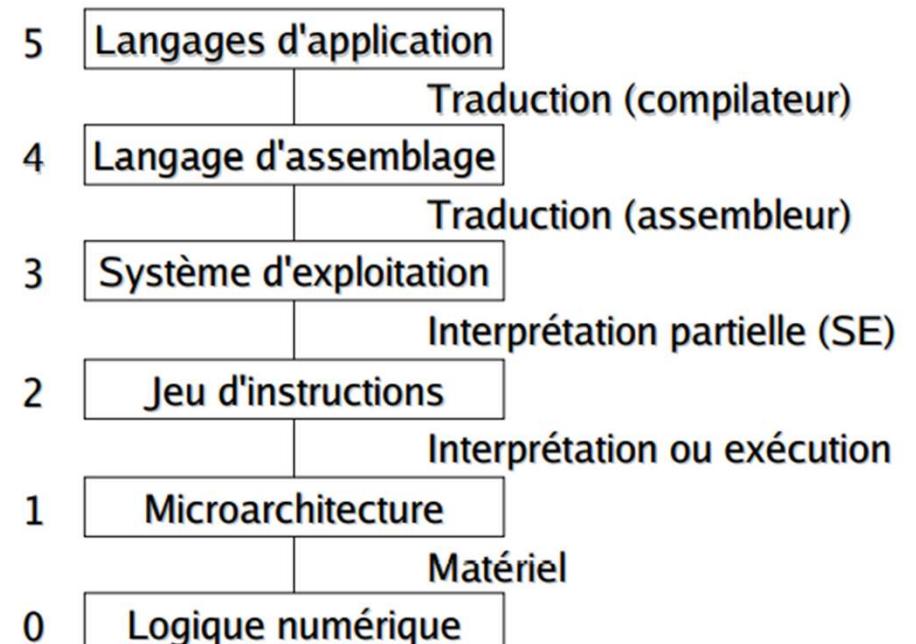
□ Couche jeu d'instruction

- La couche de l'architecture du jeu d'instructions (« Instruction Set Architecture ou « ISA ») est définie par le jeu des instructions disponibles sur la machine
- Ces instructions peuvent être exécutées par microprogramme ou bien directement



□ Couche système d'exploitation

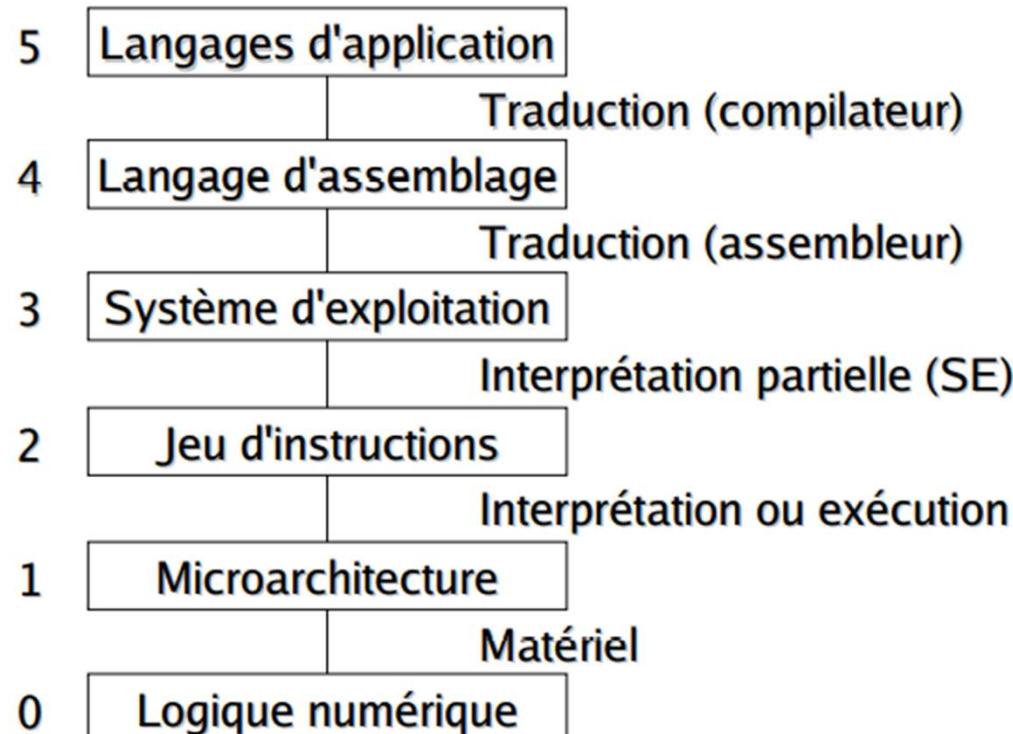
- Cette couche permet de bénéficier des services offerts par le système d'exploitation
 - Organisation mémoire, exécution concurrente, etc.
- La plupart des instructions disponibles à ce niveau sont directement traitées par les couches inférieures
- Les instructions spécifiques au système font l'objet d'une interprétation partielle (appels système)



Couches ordinateur

□ Couche langage d'assemblage

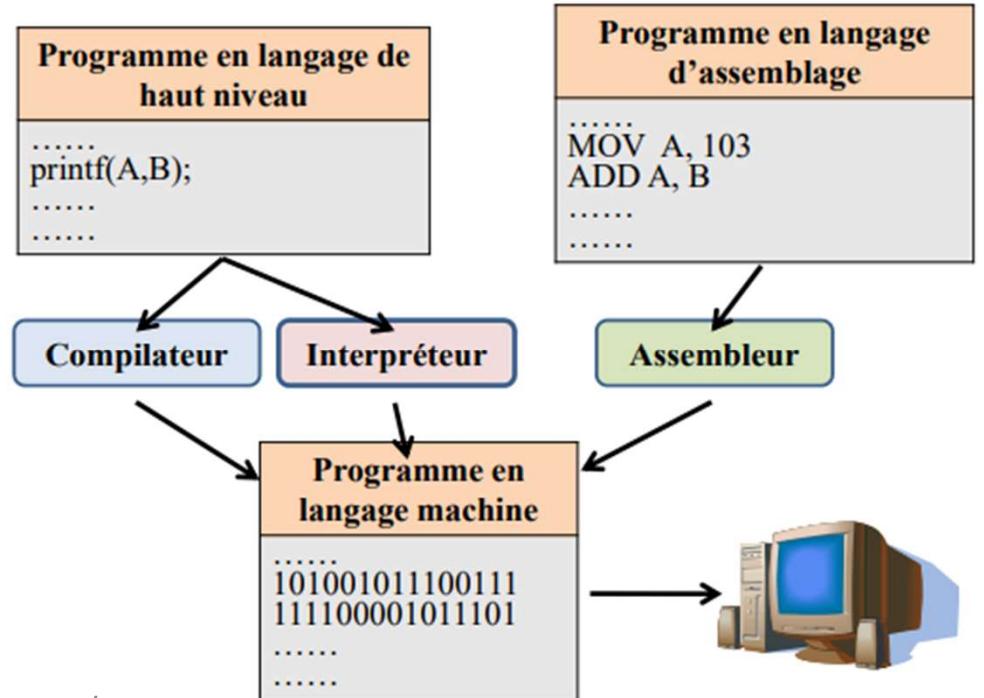
- Offre une forme symbolique aux langages des couches inférieures
- Permet à des humains d'interagir avec les couches inférieures



□ Couche langage d'application

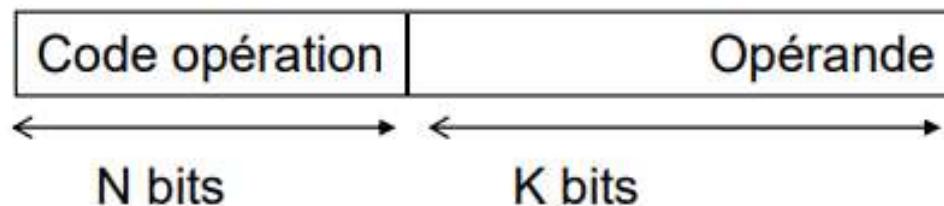
- Met à la disposition des programmeurs d'applications un ensemble de langages adaptés à leurs besoins
- Langages dits « de haut niveau »

Programmation



Codage d'une instruction

- Les **instructions et leurs opérandes** (données) sont stocké dans la mémoire.
- La taille d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type de l'instruction et du type de l'opérande.
- L'instruction est découpée en deux parties :
 - **Code opération** (code instruction) : un code sur N bits qui indique quelle instruction.
 - **La champs opérande** : qui contient la donnée ou la référence (adresse) à la donnée.



- Le format d'une instruction peut ne pas être le même pour toutes les instructions.
- Le champs opérande peut être découpé à son tour en **plusieurs champs**



Codage d'une instruction

Machine à 3 adresses

- Dans ce type de machine pour chaque instruction il faut préciser :
 - l'adresse du premier opérande
 - du deuxième opérande
 - et l'emplacement du résultat

Code opération	Opérande1	Opérande2	Résultat
----------------	-----------	-----------	----------

Exemple :

ADD A,B,C (C←B+A)

- Dans ce type de machine la taille de l'instruction est grande .
- Pratiquement il n'existent pas de machine de ce type.



Codage d'une instruction

Machine à 2 adresses

- Dans de type de machine pour chaque instruction il faut préciser :
 - l'adresse du premier opérande
 - du deuxième opérande ,
- l'adresse de résultat est implicitement l'adresse du deuxième opérande .

Code opération	Opérande1	Opérande2
----------------	-----------	-----------

Exemple :

ADD A,B

(B←A +B)



Codage d'une instruction

Machine à 1 adresses

- Dans ce type de machine pour chaque instruction il faut préciser uniquement l'adresse du **deuxième opérande**.
- Le **premier opérande** existe dans le registre **accumulateur**.
- Le **résultat** est mis dans le **registre accumulateur**.

Code opération	Opérande2
----------------	-----------

Exemple :

ADD A (ACC \leftarrow (ACC) + A)

Ce type de machine est le plus utilisé.

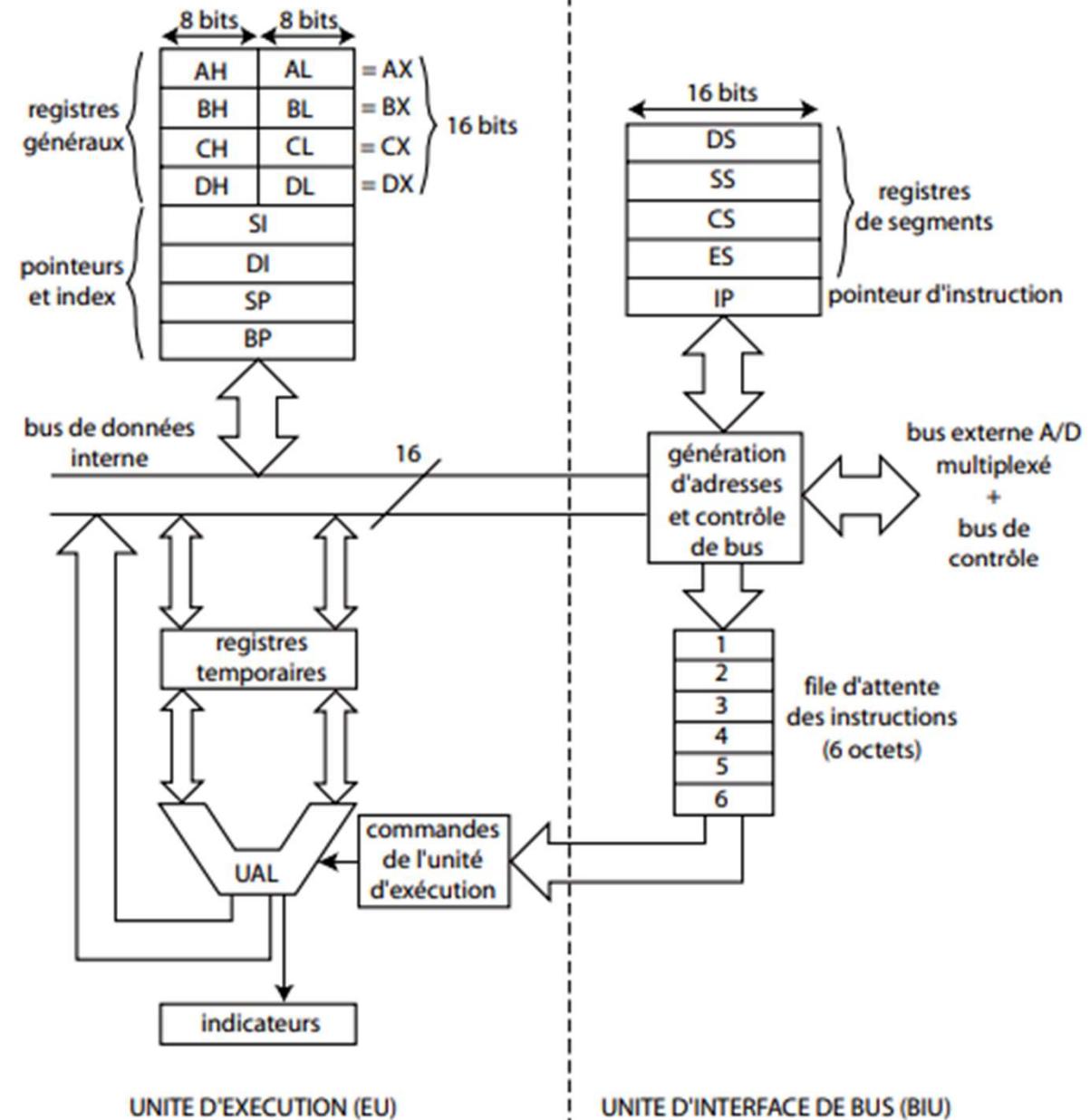
Registres sur 8086

■ Le 8086 est constitué de deux unités fonctionnant en parallèle :

- L'unité d'exécution (EU : Execution Unit)
- L'unité d'interface de bus (BIU : Bus Interface Unit)

■ **Rôle des deux unités :**

- L'unité d'interface de bus (BIU) : recherche les instructions en mémoire et les range dans une file d'attente ;
- L'unité d'exécution (EU) : exécute les instructions contenues dans la file d'attente. Les deux unités fonctionnent simultanément, d'où une accélération du processus d'exécution d'un programme (fonctionnement selon le principe du pipe-line).





Registres sur 8086

- Le microprocesseur **8086** contient **14 registres répartis en 4 groupes** :
 - **Registres généraux : 4 registres sur 16 bits**
 - **AX = (AH,AL) ; BX = (BH,BL) ; CX = (CH,CL) ; DX = (DH,DH).**
 - Ils peuvent être également considérés comme **8 registres sur 8 bits**.
 - Ils servent à contenir temporairement des données. Ce sont des registres généraux mais ils peuvent être utilisés pour des opérations particulières.
Exemple : AX = accumulateur, CX = compteur, BX: base.

■ **Registres de pointeurs et d'index : 4 registres sur 16 bits**

Pointeurs :

SP : Stack Pointer, pointeur de pile (la pile est une zone de sauvegarde de données en cours d'exécution d'un programme) ;

BP : Base Pointer, pointeur de base, utilisé pour adresser des données sur la pile.

Index :

SI : Source Index ;

DI : Destination Index.

Ils sont utilisés pour les transferts de chaînes d'octets entre deux zones mémoire.

Registres sur 8086

■ Pointeur d'instruction (IP) et indicateurs (flags) : 2 registres sur 16 bits.

- IP : Le pointeur d'instruction contient l'adresse de la prochaine instruction à exécuter

- Flags : Flags :

				O	D	I	T	S	Z	A		P		C	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CF : indicateur de retenue (carry) ;

SF : indicateur de signe ;

PF : indicateur de parité ;

TF : indicateur d'exécution pas à pas (trap) ;

AF : indicateur de retenue auxiliaire ;

IF : indicateur d'autorisation d'interruption ;

ZF : indicateur de zéro ;

DF : indicateur de décrémentation ;

OF : indicateur de dépassement (overflow).

Les instructions de branchement conditionnels utilisent les *indicateurs*, qui sont des bits spéciaux positionnés par l'UAL après certaines opérations. Les indicateurs sont regroupés dans le *registre d'état* du processeur. Ce registre n'est pas accessible globalement par des instructions ; chaque indicateur est manipulé individuellement par des instructions spécifiques.

Nous étudierons ici les indicateurs nommés ZF, CF, SF et OF.

□ Indicateurs flags

ZF *Zero Flag*

Cet indicateur est mis à 1 lorsque le résultat de la dernière opération est zéro. Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF=1 indique que les deux opérandes étaient égaux. Sinon, ZF est positionné à 0.

CF *Carry Flag*

C'est l'indicateur de report (retenue), qui intervient dans les opérations d'addition et de soustractions sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP.

CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes. Exemples (sur 4 bits pour simplifier) :

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \\ + 0 \ 1 \ 1 \ 0 \\ \hline \end{array}$$

CF=0 1 0 1 0

$$\begin{array}{r} 1 \ 1 \ 0 \ 0 \\ + 0 \ 1 \ 1 \ 0 \\ \hline \end{array}$$

CF=1 0 0 1 0

$$\begin{array}{r} 1 \ 1 \ 1 \ 1 \\ + 0 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

CF=1 0 0 0 0

Registres sur 8086

Indicateurs flags

SF *Sign Flag*

SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0. SF est utile lorsque l'on manipule des entiers relatifs, car le bit de poids fort donne alors le signe du résultat. Exemples (sur 4 bits) :

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \\ + \ 0 \ 1 \ 1 \ 0 \\ \hline \end{array} \quad \begin{array}{r} 1 \ 1 \ 0 \ 0 \\ + \ 0 \ 1 \ 1 \ 0 \\ \hline \end{array} \quad \begin{array}{r} 1 \ 1 \ 1 \ 1 \\ + \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

SF=1 1 0 1 0 SF=0 0 0 1 0 SF=0 0 0 0 0

OF *Overflow Flag*

Indicateur de débordement⁴ OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable *en relatif* dans l'accumulateur (par exemple si l'addition de 2 nombres positifs donne un codage négatif).

$$\begin{array}{r} 0 \ 1 \ 0 \ 0 \\ + \ 0 \ 1 \ 1 \ 0 \\ \hline \end{array} \quad \begin{array}{r} 1 \ 1 \ 0 \ 0 \\ + \ 0 \ 1 \ 1 \ 0 \\ \hline \end{array} \quad \begin{array}{r} 1 \ 1 \ 1 \ 1 \\ + \ 0 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

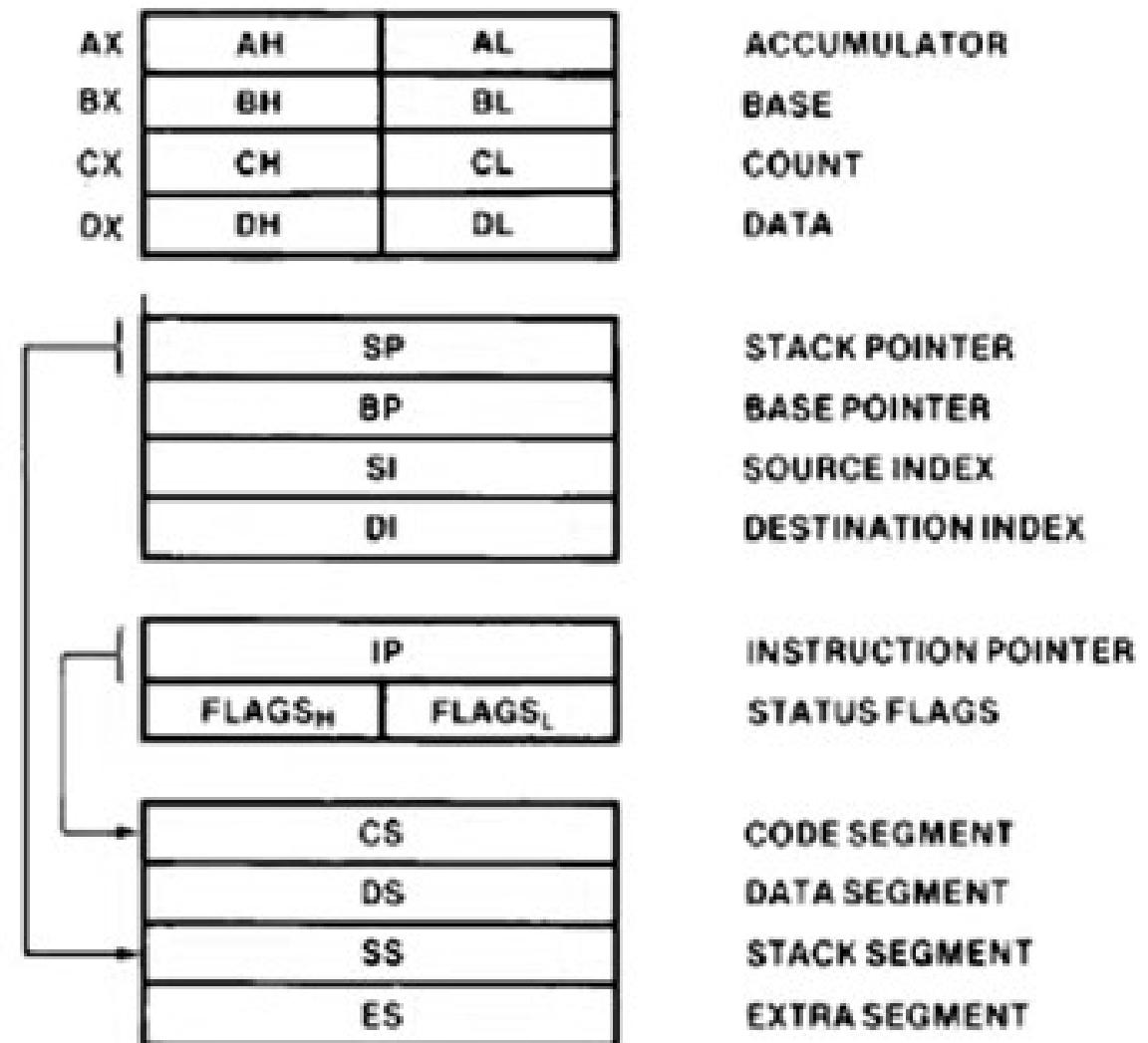
OF=1 1 0 1 0 OF=0 0 0 1 0 OF=0 0 0 0 0

Lorsque l'UAL effectue une addition, une soustraction ou une comparaison, les quatre indicateurs sont positionnés. Certaines autres instructions que nous étudierons plus loin peuvent modifier les indicateurs.

Registres sur 8086

- **Registres de segments:**

- 4 registres sur 16 bits qui permettent d'indiquer **les positions de 4 segments**.
- Registres de segments : 4 registres sur **16 bits**.
 - **CS : Code Segment**, registre de segment de code ;
 - **DS : Data Segment**, registre de segment de données ;
 - **SS : Stack Segment**, registre de segment de pile ;
 - **ES : Extra Segment**, registre de segment supplémentaire pour les données ;
- Les registres de segments, associés aux pointeurs et aux index, permettent au microprocesseur 8086 d'adresser l'ensemble de la mémoire.



□ Segmentation de la mémoire

L'espace mémoire adressable par le 8086 est de $2^{20} = 1\ 048\ 576$ octets = 1 Mo (20 bits d'adresses). Cet espace est divisé en **segments**. Un segment est une zone mémoire de 64 Ko (65 536 octets) définie par son adresse de départ qui doit être un multiple de 16. Dans une telle adresse, les 4 bits de poids faible sont à zéro. On peut donc représenter l'adresse d'un segment avec seulement ses 16 bits de poids fort, les 4 bits de poids faible étant implicitement à 0.

Pour désigner une case mémoire parmi les $2^{16} = 65\ 536$ contenues dans un segment, il suffit d'une valeur sur 16 bits.

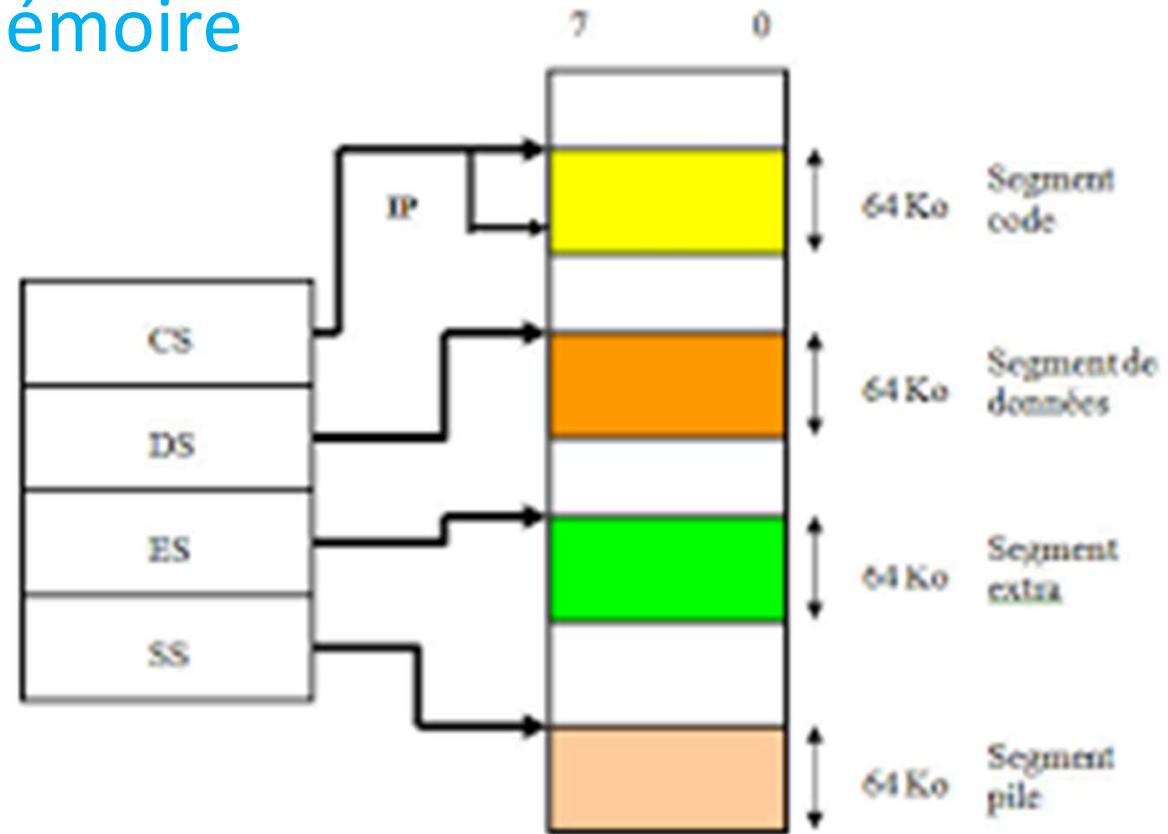
Ainsi, une case mémoire est repérée par le 8086 au moyen de deux quantités sur 16 bits :

- l'adresse d'un segment ;
- un déplacement ou **offset** (appelé aussi **adresse effective**) dans ce segment.

Cette méthode de gestion de la mémoire est appelée **segmentation de la mémoire**.

□ Segmentation de la mémoire

- Le fait d'injecter 4 zéros en poids faible du segment revient à effectuer un **décalage de 4 positions vers la gauche**, c'est à dire **une multiplication par $2^4 = 16$** .
- A un instant donné, le 8086 a accès à 4 segments dont les adresses se trouvent dans les registres de segment CS, DS, SS et ES.
- Le segment de code contient les **instructions du programme**, le segment de données contient les **données manipulées par le programme**, le segment de pile contient la **pile de sauvegarde** et le segment supplémentaire peut aussi contenir **des données**.



Division de la mémoire en segments

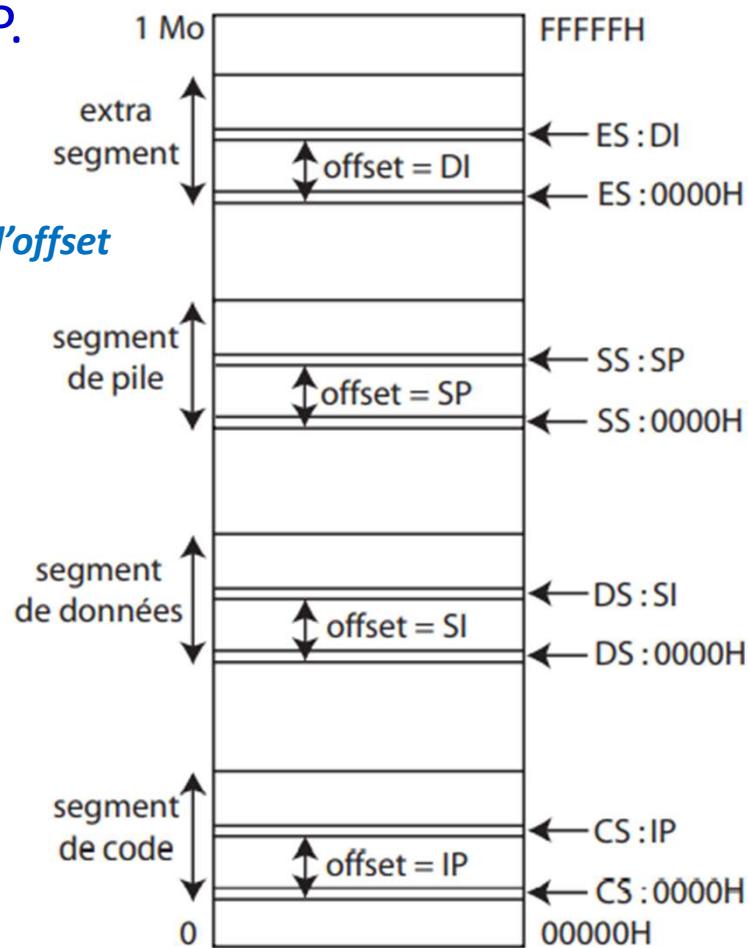
- Le registre **CS** est associé au pointeur d'instruction **IP**, ainsi la prochaine instruction à exécuter se trouve à l'adresse logique **CS : IP**.

□ Segmentation de la mémoire

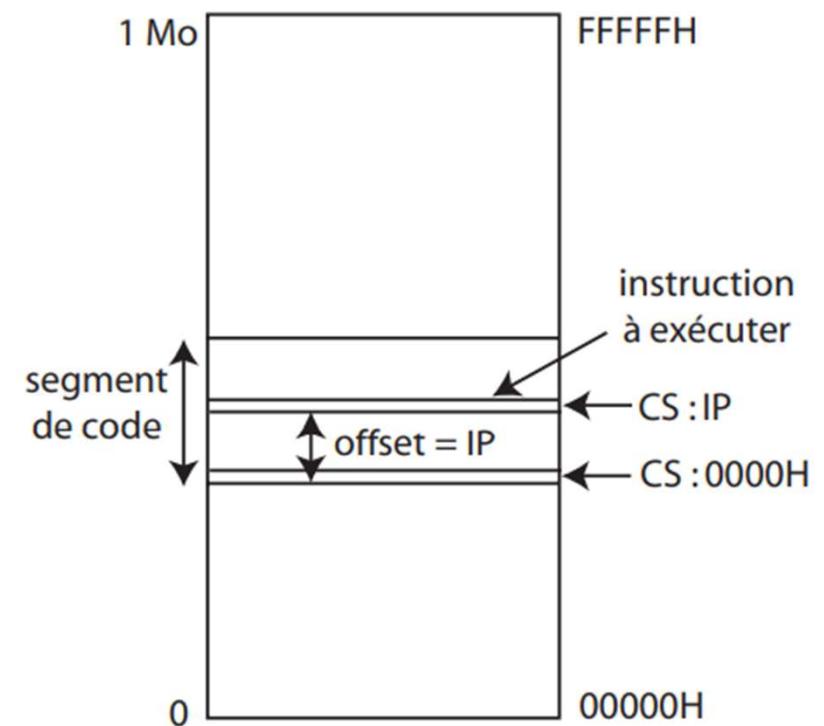
- De même, les registres de **segments DS et ES** peuvent être associées à un **registre d'index**. Exemple : DS : SI, ES : DI.
- Le registre de **segment de pile SS** peut être associé aux **registres de pointeurs** : SS : SP ou SS : BP.

Association :

registre segment-registre d'offset



■ Mémoire accessible par le 8086 à un instant donné :





Gestion de la mémoire par le 8086

□ Segmentation de la mémoire

Contenu des registres après un RESET du microprocesseur :

IP = 0000H

CS = FFFFH

DS = 0000H

ES = 0000H

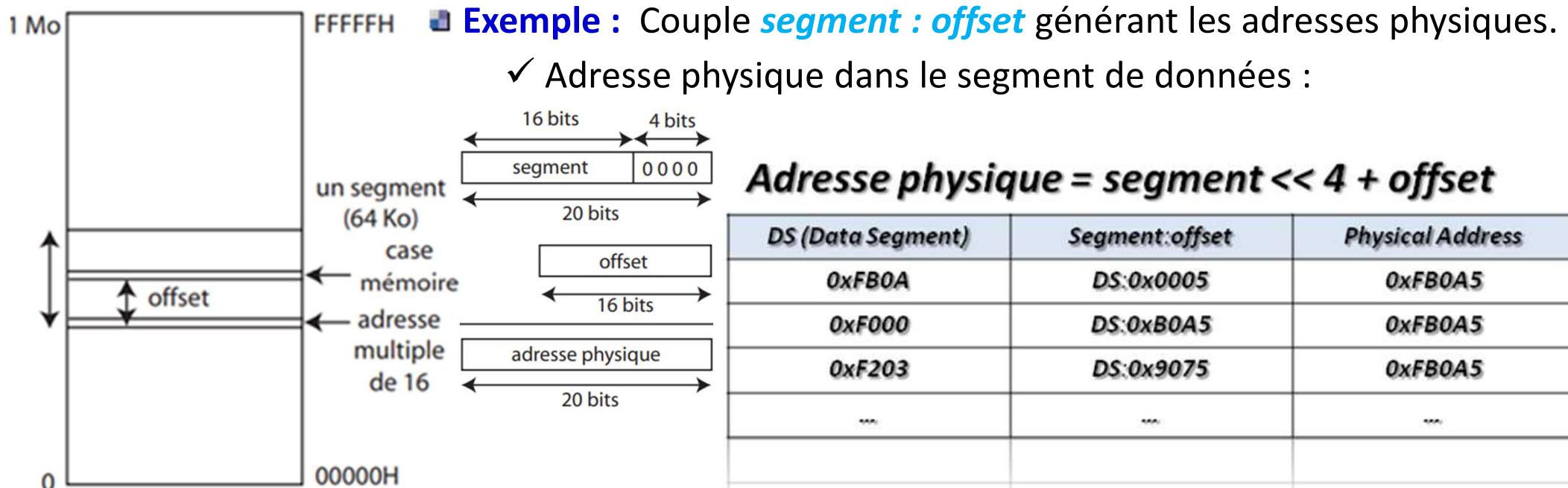
SS = 0000H

Puisque CS contient la valeur FFFFH et IP la valeur 0000H, la première instruction exécutée par le 8086 se trouve donc à l'adresse logique FFFFH : 0000H, correspondant à l'adresse physique FFFF0H (bootstrap). Cette instruction est généralement un saut vers le programme principal qui initialise ensuite les autres registres de segment.

□ Correspondance entre adresse logique et adresse physique

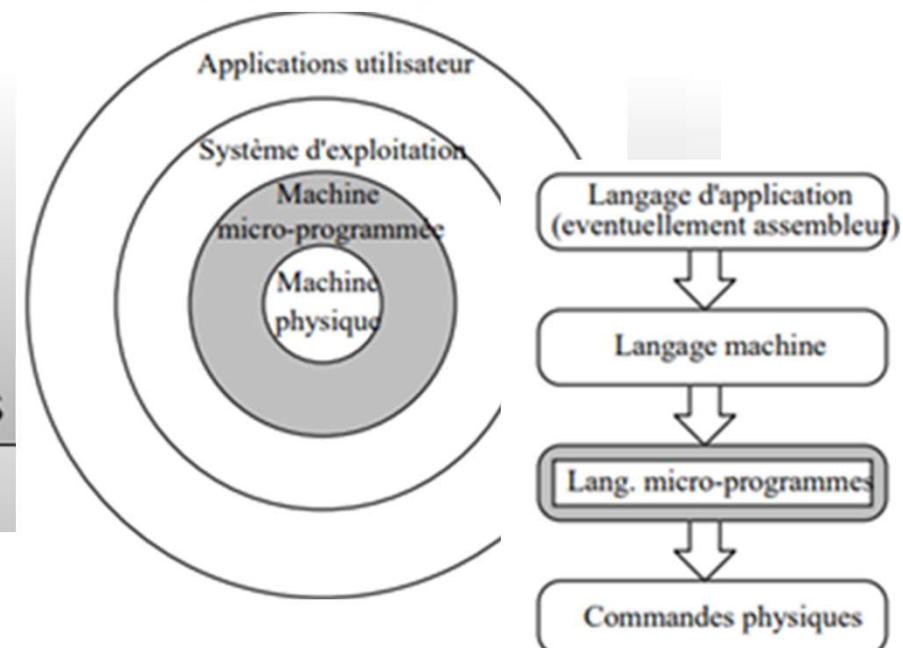
La donnée d'un couple (segment,offset) définit une **adresse logique**, notée sous la forme **segment : offset**.

L'adresse d'une case mémoire donnée sous la forme d'une quantité sur 20 bits (5 digits hexa) est appelée **adresse physique** car elle correspond à la valeur envoyée réellement sur le bus d'adresses A0 - A19.



□ Le microprocesseur 8086

- Disponible depuis juillet 1978
- Le premier microprocesseur 16 bits développé par Intel.
- Le premier de la famille 80x86.
- Constitué de 29000 transistors sur une puce de 32.7 mm².
- Existe en trois versions : 8086 (5 MHz), 8086-2 (8 MHz) et le 8086-1 (10 MHz).
- Peut fonctionner dans des systèmes mono ou bien multiprocesseurs.
- Caractéristiques :
 - Adressage direct de 1MO ;
 - 14 registres de 16 bits ;
 - 24 modes d'adresses ;
 - Opérations sur 1, 8 et 16 bits ;
 - Opérations arithmétiques sur des opérandes signés ou non signés de 8 et 16 bits





Le 8086

□ Pourquoi l'assembleur

- Ecriture des instructions en langage symbolique :
- Voici un programme en **langage machine 80486**, implanté à l'adresse 0100H :
A1 01 10 03 06 01 12 A3 01 14.
- Il additionne le contenu de 2 cases mémoire et range le résultat dans une 3ème.
- Nous avons simplement transcrit en hexadécimal le code du programme.
- Il est clair que ce type d'écriture n'est pas très utilisable par un être humain. A chaque instruction que peut exécuter le processeur correspond une représentation binaire sur un ou plusieurs octets, comme on l'a vu plus haut.
- C'est le travail du processeur de décoder cette représentation pour effectuer les opérations correspondantes. Afin de pouvoir écrire (et relire) des programmes en langage machine, on utilise une notation symbolique, appelée **langage assembleur**.

Adresse	Contenu MP	Langage Symbolique	Explication en français
0100	A1 01 10	MOV AX, [0110]	Charger AX avec le contenu de 0110.
0103	03 06 01 12	ADD AX, [0112]	Ajouter le contenu de 0112 à AX (résultat dans AX).
0107	A3 01 14	MOV [0114], AX	Ranger AX en 0114.



Le 8086

□ Pourquoi l'assembleur

- Ainsi, la première instruction du programme ci-dessus (code A1 01 10) sera notée : MOV AX, [0110] elle indique que le mot mémoire d'adresse 0110H est chargé dans le registre AX du processeur.
- On utilise des programmes spéciaux, appelés assembleurs, pour traduire automatiquement le langage symbolique en code machine. La transcription langage symbolique du programme complet est donnée ci-dessus. L'adresse de début de chaque instruction est indiquée à gauche (en hexadécimal).
- Lorsque l'on doit lire ou écrire un programme en langage machine, il est difficile d'utiliser la notation hexadécimale. On écrit les programmes à l'aide de symboles comme MOV, ADD, etc.
- Les concepteurs de processeur, comme Intel, fournissent toujours une documentation avec les **codes des instructions** de leur processeur, et les symboles correspondant.
- Debug est un programme, très utile pour traduire automatiquement les symboles des instructions en code machine. Cependant, debug n'est utilisable que pour mettre au point de petits programmes. En effet, le programmeur doit spécifier lui même les adresses des données et des instructions.



Le 8086

□ Pourquoi l'assembleur

- ◆ Soit par exemple le programme suivant, qui multiplie une donnée en mémoire par 8 :

```
0100    MOV BX, [0112] ; charge la donnee
0103    MOV AX, 3
0106    SHL BX          ; decale a gauche
0108    DEC AX
0109    JNE 0106         ; recommence 3 fois
010B    MOV [0111], BX  ; range le resultat
010E    MOV AH, 4C
0110    INT 21H
0112    ; on range ici la donnee
```

- ▶ Nous avons spécifié que la donnée était rangée à l'adresse 0111H, et que l'instruction de branchement JNE allait en 0106H.
- ▶ Si l'on désire modifier légèrement ce programme, par exemple ajouter une instruction avant MOV BX, [0112], il va falloir modifier ces deux adresses (de saut: 0106 et de rangement: 0111). On conçoit aisément que ce travail devienne très difficile si le programme manipule beaucoup de variables.



Modes d'adressage

- La champs opérande contient **la donnée** ou **la référence** (**adresse**) à la donnée.
- Le mode d'adressage définit la manière dont le microprocesseur va **accéder à l'opérande**.
- Le code opération de l'instruction comportent un ensemble de bits pour indiquer le **mode d'adressage**.
- Les modes d'adressage les plus utilisés sont :
 - Immédiat
 - Direct
 - Indirect
 - Indexé
 - relatif
- Selon la manière dont la donnée est spécifiée, c'est à dire selon le mode d'adressage de la donnée, une instruction sera codée par 1, 2, 3 ou 4 octets.



Modes d'adressage

□ Adressage implicite

L'instruction contient seulement le code opération, sur 1 ou 2 octets.

code opération
(1 ou 2 octets)

L'instruction porte sur des registres ou spécifie une opération sans opérande (exemple : “incrémenter AX”).

□ Adressage immédiat (1)

- L'opérande existent dans le **champs adresse** de l'instruction

Code opération	Opérande
----------------	----------

Exemple :

ADD 150

ADD	150
-----	-----

Cette commande va avoir l'effet suivant : ACC←(ACC)+ 150

Si le registre accumulateur contient la valeur 200 alors
après l'exécution son contenu sera égale à 350

Modes d'adressage

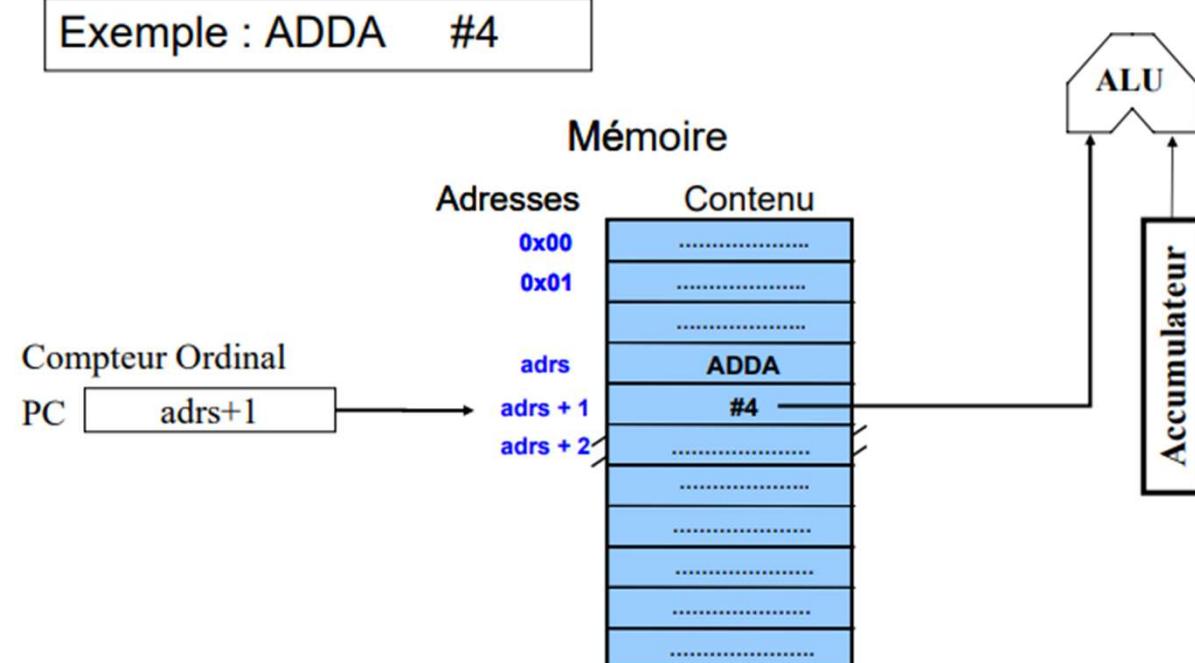
□ Adressage immédiat (2)

Le champ opérande contient la donnée (une valeur constante sur 1 ou 2 octets).

code opération (1 ou 2 octets)	valeur (1 ou 2 octets)
-----------------------------------	---------------------------

Exemple : “Ajouter la valeur 5 à AX”. Ici l’opérande 5 est codée sur 2 octets puisque l’opération porte sur un registre 16 bits (AX).

Exemple : ADDA #4



- *Exemple 3 : MOV AX, 12*

Modes d'adressage

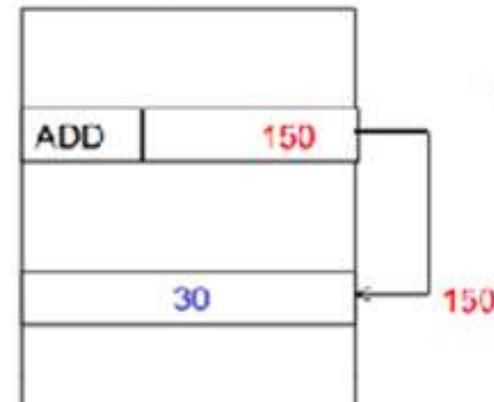
□ Adressage direct (1)

- Le champs opérande de l'instruction contient **l'adresse de l'opérande** (emplacement en mémoire)
- Pour réaliser l'opération il faut le récupérer (lire) l'opérande à partir de la mémoire. $ACC \leftarrow (ACC) + (ADR)$

Exemple :

On suppose que l'accumulateur
contient la valeur 20 .

A la fin de l'exécution nous
allons avoir la valeur 50 (20 + 30)



Le champ opérande contient **l'adresse de la donnée** en mémoire principale sur 2 octets.

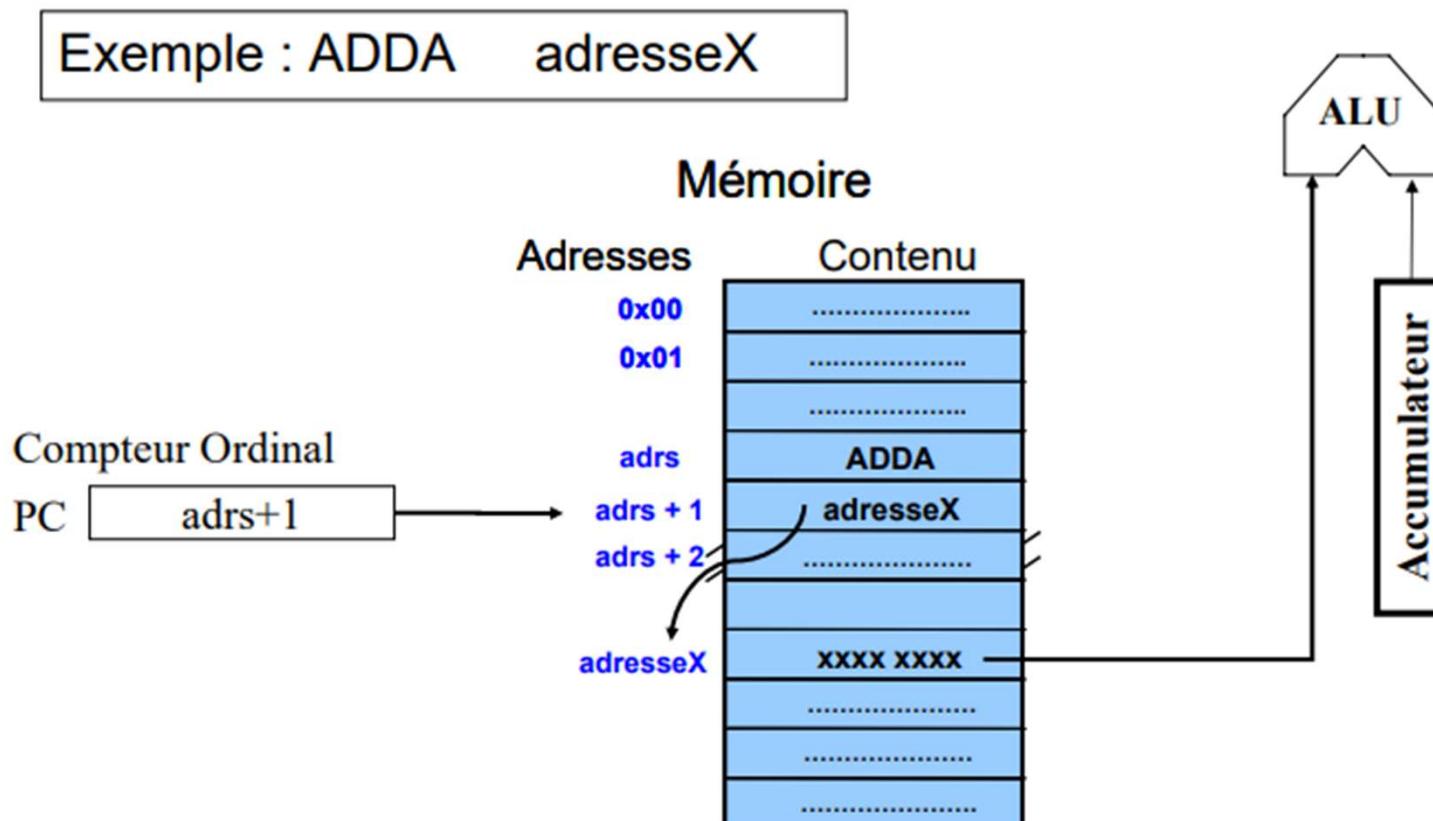
code opération (1 ou 2 octets)	adresse de la donnée (2 octets)
-----------------------------------	------------------------------------

Attention : dans le 80x86, les adresses sont toujours manipulées sur 16 bits, quelle que soit la taille réelle du bus.

Exemple : "Placer dans AX la valeur contenue à l'adresse 130H".

Modes d'adressage

□ Adressage direct (2)



Modes d'adressage

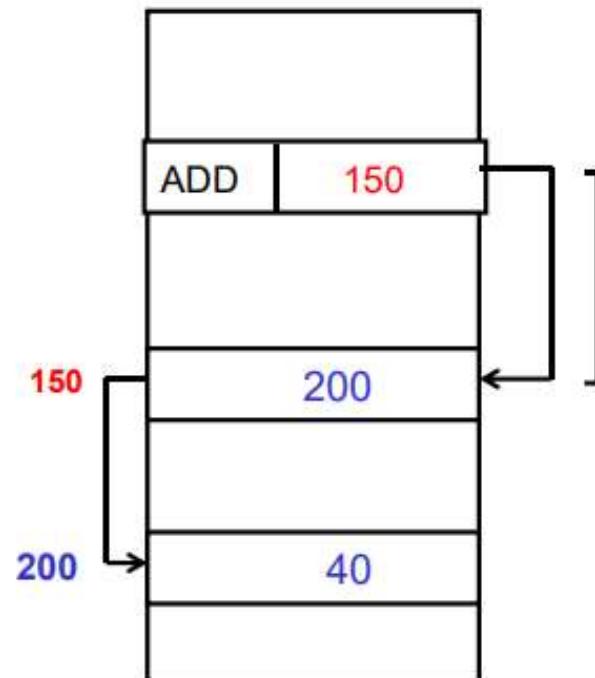
□ Adressage indirect (1)

- La champs adresse contient **l'adresse de l'adresse** de l'opérande.
- Pour réaliser l'opération il faut :
 - Récupérer **l'adresse de l'opérande** à partir de la mémoire.
 - Par la suite il faut chercher l'opérande à partir de la mémoire.

$$ACC \leftarrow (ACC) + ((ADR))$$

- Exemple :
- Initialement l'accumulateur contient la valeur 20
- Il faut récupérer l'adresse de l'adresse (150).
- Récupérer l'adresse de l'opérande à partir de l'adresse **150** (la valeur 200)
- Récupérer la valeur de l'opérande à partir de l'adresse 200 (**la valeur 40**)

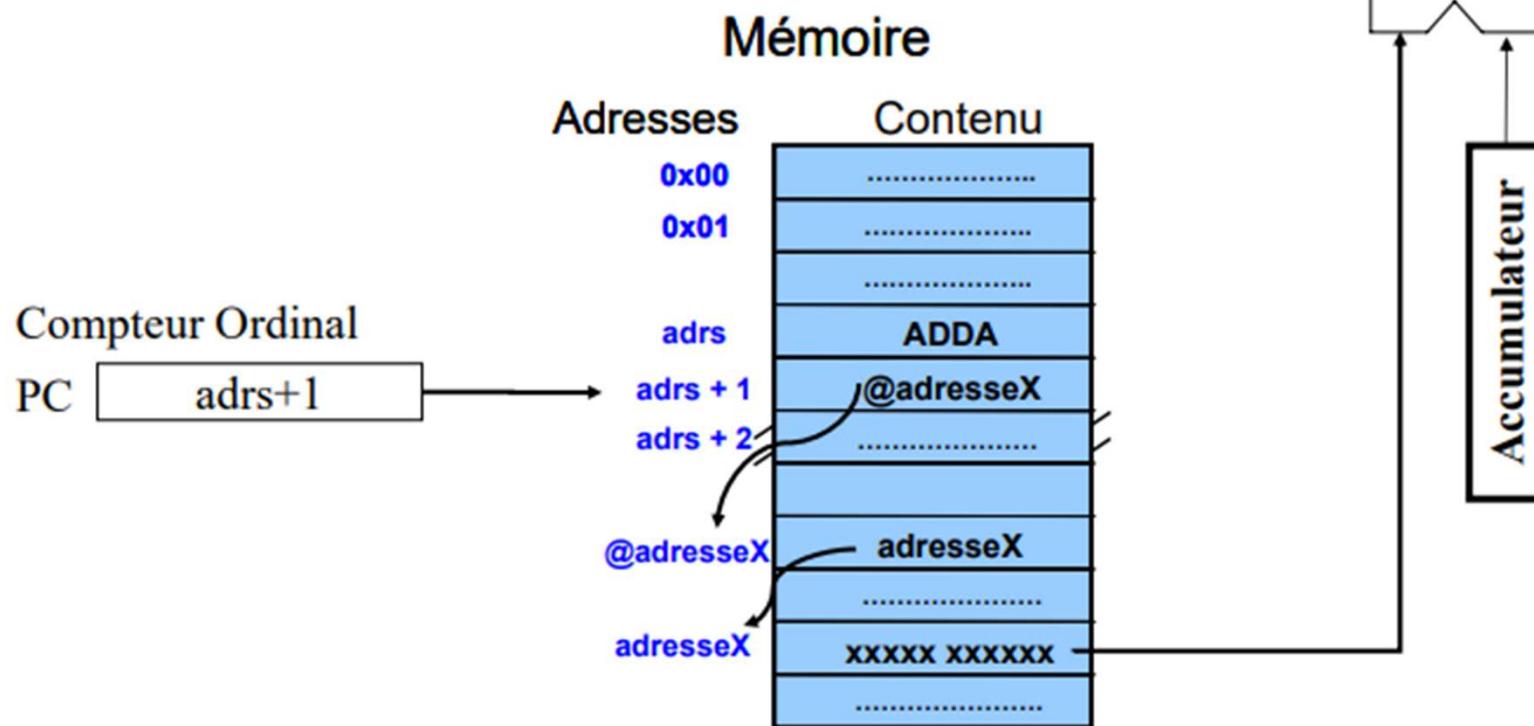
Additionner la valeur 40 avec le contenu de l'accumulateur (20) et nous allons avoir la valeur **60**



Modes d'adressage

□ Adressage indirect (2)

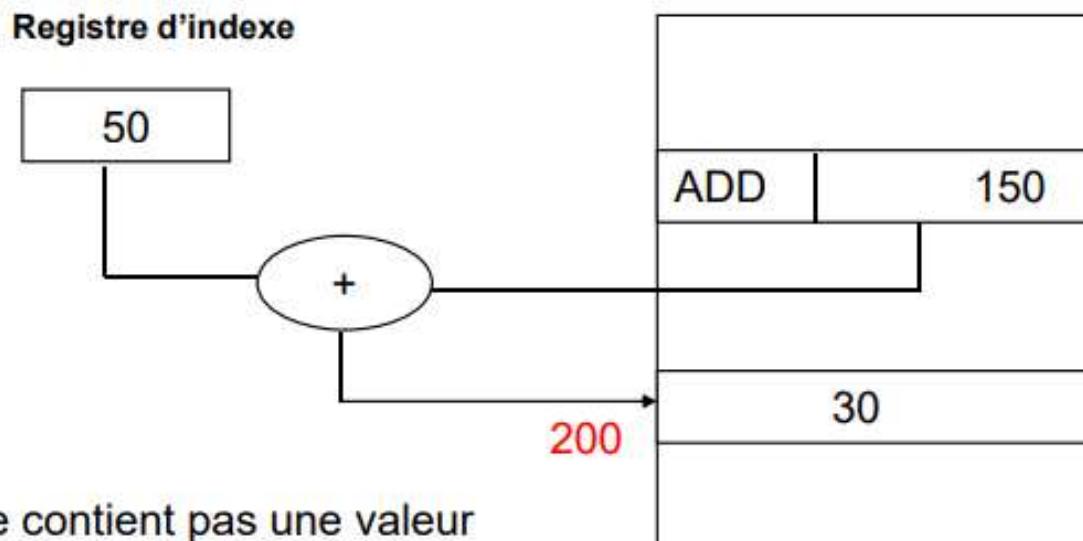
Exemple : ADDA @adresseX



Modes d'adressage

□ Adressage indexé DI et SI

- L'adresse effectif de l'opérande est relatif à une zone mémoire.
- L'adresse de cette zone se trouve dans un registre spécial (registre indexé).
- Adresse opérande = ADR + (X)



Remarque : si ADR ne contient pas une valeur immédiate alors

$$\text{Adresse opérande} = (\text{ADR}) + (X)$$



Modes d'adressage

□ Adressage relatif

Ce mode d'adressage est utilisé pour certaines instructions de branchement. Le champ opérande contient un entier relatif codé sur 1 octet, nommé *déplacement*, qui sera ajouté à la valeur courante de IP.

code opération (1 octet)	déplacement (1 octet)
-----------------------------	--------------------------

En adressage relatif, on indique simplement l'adresse (hexa). L'assembleur traduit automatiquement cette adresse en un déplacement (relatif sur un octet). Exemple :

JNE 0108

(nous étudierons l'instruction JNE plus loin).



Modes d'adressage

□ Autres types d'adressage

➤ D'autres types d'adressage existent.

■ **Adressage registre à registre:** Les 2 opérandes se trouvent dans des registres. Exemples : **MOV AX, BX** ; opérandes 16 bits

ADD CH, DL ; opérandes 8 bits

■ **Adressage basé (par registre) :** Il existe deux registres de base: le registre **BX** et le registre **BP** :

- Le registre **BX** est utilisé avec l'association de l'un des **segments des données** qui sont: **DS et ES**. Exemples : **MOV AL, [BX]**
ou bien **MOV AL, DS : [BX]**.

(ils donnent le même résultat).

MOV AL, ES : [BX]

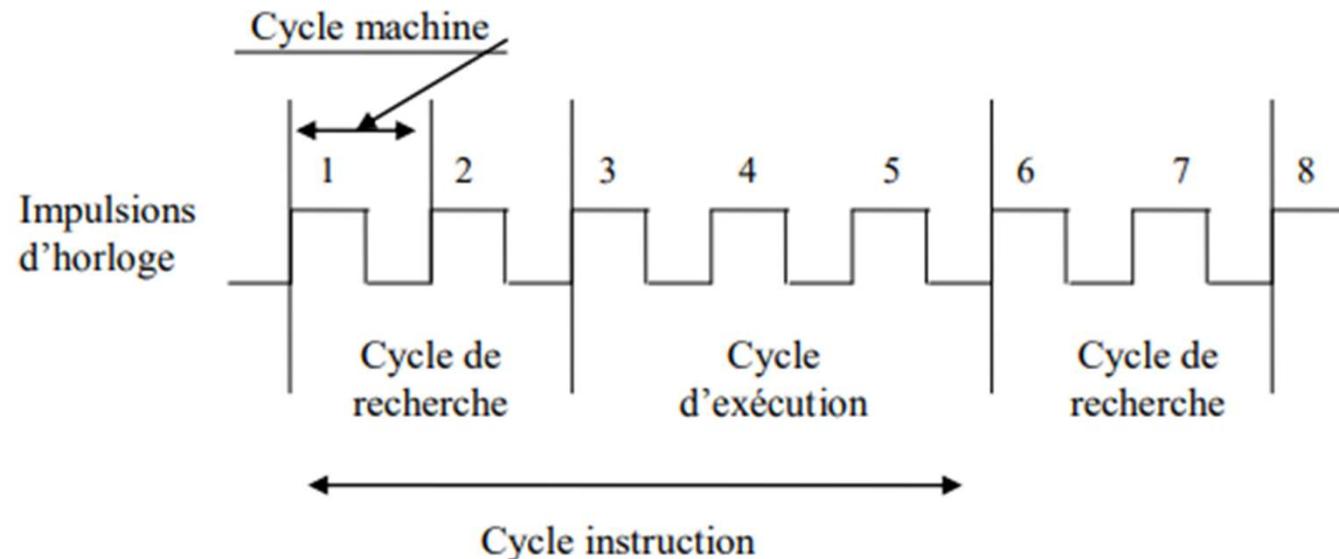
- Le registre **BP** est associé à la **pile**, par exemple : **MOV AL, [BP]** va transférer une donnée de la pile à partir d'une position pointée par le contenu du registre [BP] vers AL.

■ **Adressage basé indexé avec ou sans déplacement :** Exemples :

MOV AL, [BX+SI]; MOV AL, [BX+DI+200h]; MOV AL, [BX+SI+200h]

Temps d'exécution d'une instruction

☐ Cycle machine versus cycle instruction



- Les signaux périodiques générés par l'horloge définissent le cycle de base ou cycle machine, durée élémentaire régissant le fonctionnement de la machine.
- Un cycle instruction est composé d'un **cycle de recherche** et d'un **cycle d'exécution** et peut s'étendre sur plusieurs cycles machine.
- Le temps d'exécution du Pentium pour la plupart des instructions est de 1 à 5 cycles d'horloge.



Temps d'exécution d'une instruction

- Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer.
- Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage : il est plus long d'accéder à la mémoire principale qu'à un registre du processeur.
- La durée d'un cycle dépend de la fréquence d'horloge de l'ordinateur. Plus l'horloge bat rapidement, plus un cycle est court et plus on exécute un grand nombre d'instructions par seconde.
- ◆ Chaque instruction est caractérisée par le nombre de périodes d'horloge (ou microcycles) qu'elle utilise (donnée fournie par le fabricant du microprocesseur).
- ◆ Exemple : horloge à 5 MHz, période $T = 1/f = 0,2 \mu\text{s}$.
- ◆ Si l'instruction s'exécute en 3 microcycles, la durée d'exécution de l'instruction est : $3 \times 0,2 = 0,6 \mu\text{s}$.
- ◆ L'horloge est constituée par un oscillateur à quartz dont les circuits peuvent être internes ou externes au microprocesseur.

□ Généralités :

- Chaque microprocesseur reconnaît un ensemble d'instructions appelé jeu d'instructions (Instruction Set) fixé par le constructeur.
- Pour les microprocesseurs classiques, le nombre d'instructions reconnues varie entre 75 et 150 (microprocesseurs **CISC** : Complex Instruction Set Computer).
- Il existe aussi des microprocesseurs dont le nombre d'instructions est très réduit (microprocesseurs **RISC** : Reduced Instruction Set Computer) : entre 10 et 30 instructions, permettant d'améliorer le temps d'exécution des programmes.
- Une instruction est définie par son code opératoire, valeur numérique binaire difficile à manipuler par l'être humain.
- On utilise donc une notation symbolique pour représenter les instructions : **les mnémoniques**.
- Un programme constitué de mnémoniques est appelé programme en assembleur.
- Les instructions peuvent être classées en groupes :
 - Instructions de **transfert de données** ;
 - Instructions **arithmétiques** ;
 - Instructions **logiques** ;
 - Instructions de **branchement** ...



Programmation assembleur

□ Les instructions de transfert

Elles permettent de déplacer des données d'une **source** vers une **destination** :

- registre vers mémoire ;
- registre vers registre ;
- mémoire vers registre.

Remarque : le microprocesseur 8086 n'autorise pas les transferts de mémoire vers mémoire (pour ce faire, il faut passer par un registre intermédiaire).

Syntaxe : **MOV destination,source**

Remarque : MOV est l'abréviation du verbe « to move » : déplacer.

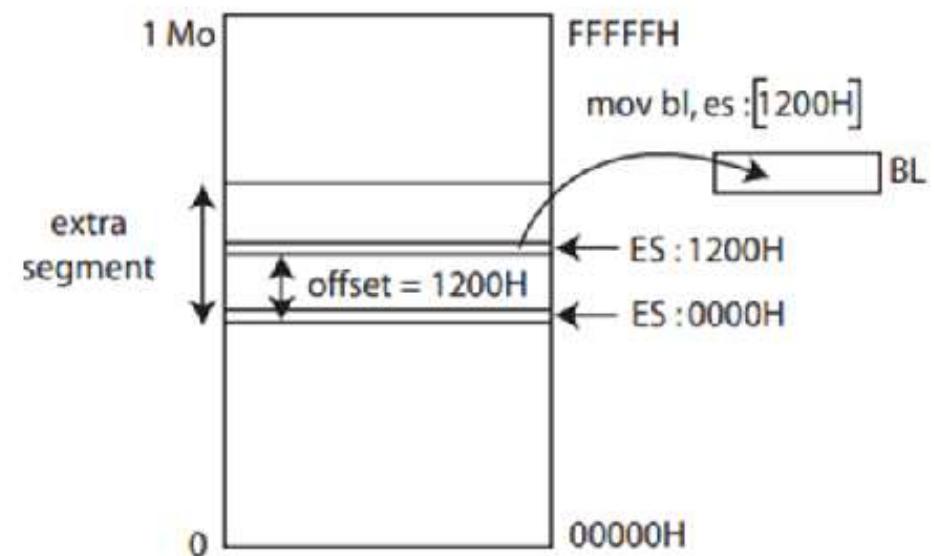
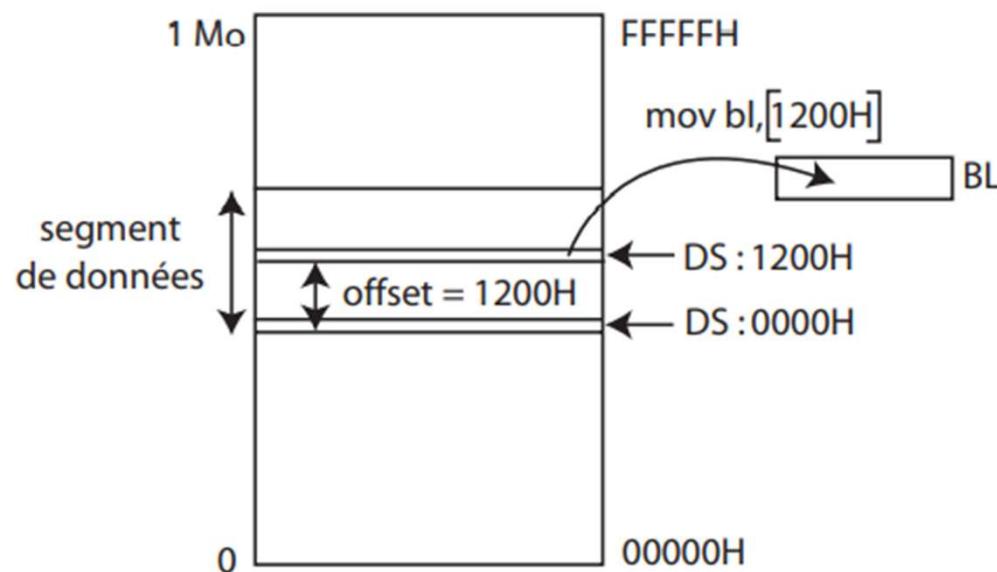
Il existe différentes façons de spécifier l'adresse d'une case mémoire dans une instruction : ce sont les **modes d'adressage**.

Exemples de modes d'adressage simples :

- **mov ax,bx** : charge le contenu du registre BX dans le registre AX. Dans ce cas, le transfert se fait de registre à registre : **adressage par registre** ;
- **mov al,12H** : charge le registre AL avec la valeur 12H. La donnée est fournie immédiatement avec l'instruction : **adressage immédiat**.

□ Les instructions de transfert

- `mov bl, [1200H]` : transfère le contenu de la case mémoire d'adresse effective (offset) 1200H vers le registre BL. L'instruction comporte l'adresse de la case mémoire où se trouve la donnée : **adressage direct**. L'adresse effective représente l'offset de la case mémoire dans le segment de données (segment dont l'adresse est contenue dans le registre DS) : segment par défaut.

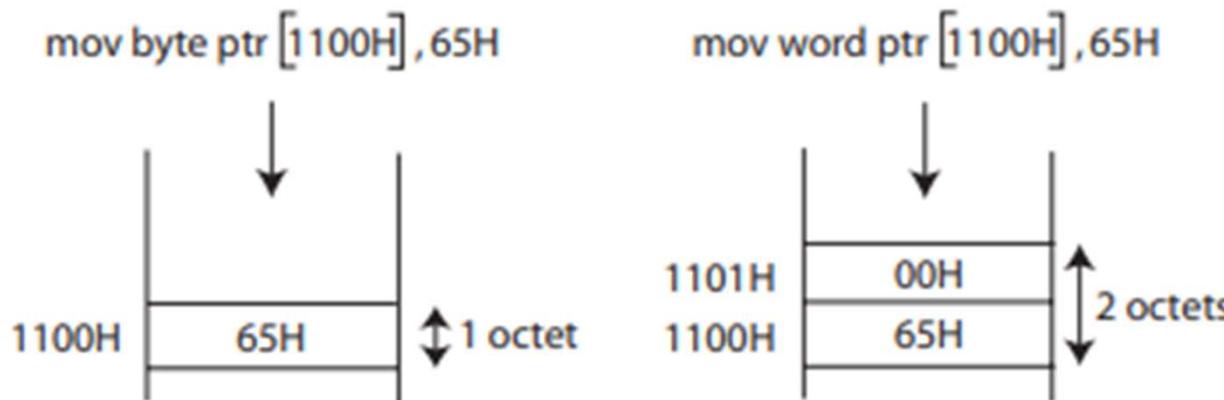


On peut changer le segment lors d'un adressage direct en ajoutant un **préfixe de segment**, exemple : `mov bl, es : [1200H]`. On parle alors de **forçage de segment**.

□ Les instructions de transfert : Remarques

Remarque : dans le cas de l'adressage immédiat de la mémoire, il faut indiquer le **format** de la donnée : octet ou mot (2 octets) car le microprocesseur 8086 peut manipuler des données sur 8 bits ou 16 bits. Pour cela, on doit utiliser un **spécificateur de format** :

- `mov byte ptr [1100H],65H` : transfère la valeur 65H (sur 1 octet) dans la case mémoire d'offset 1100H ;
- `mov word ptr [1100H],65H` : transfère la valeur 0065H (sur 2 octets) dans les cases mémoire d'offset 1100H et 1101H.



Remarque : les microprocesseurs Intel rangent l'octet de poids faible d'une donnée sur plusieurs octets à l'adresse la plus basse (**format Little Endian**).

Programmation assembleur

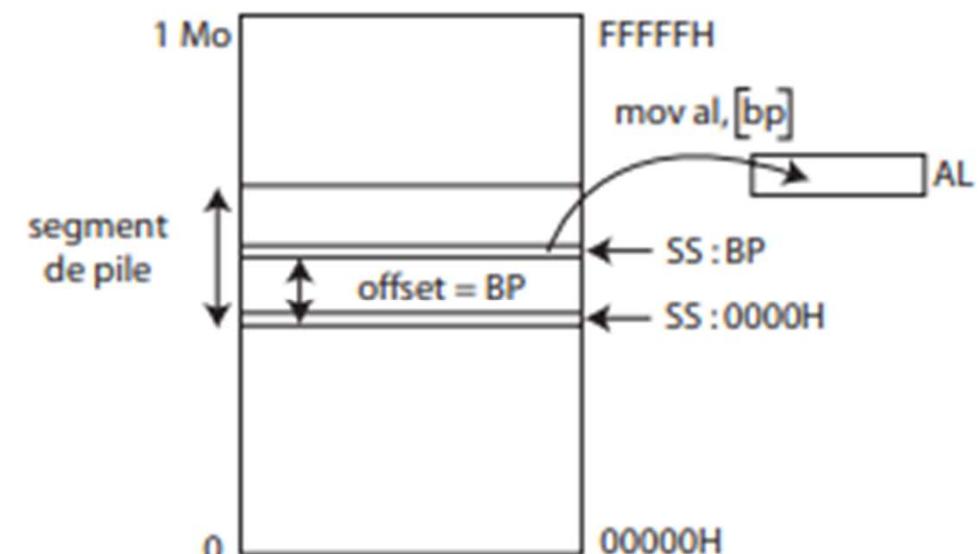
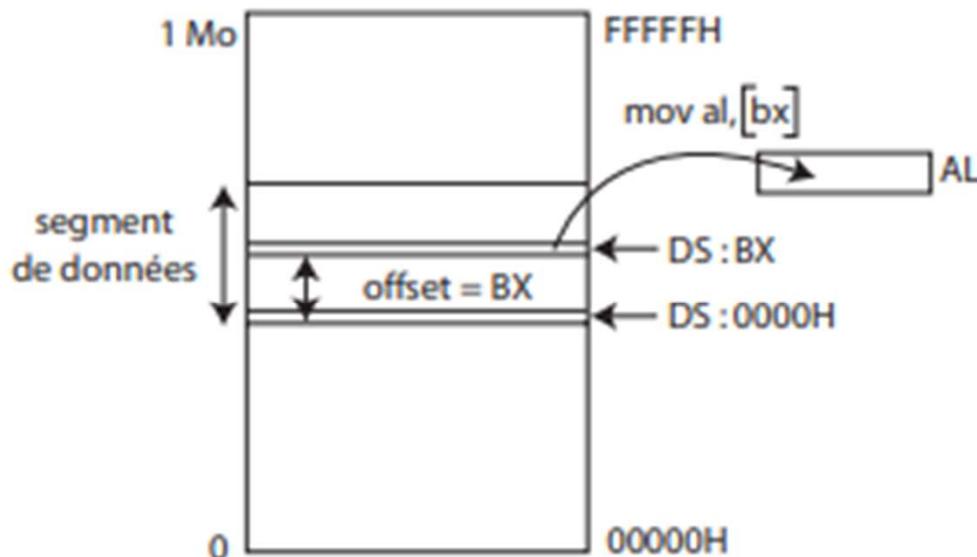
□ Les instructions de transfert

- **adressage basé** : l'offset est contenu dans un **registre de base BX ou BP**.

Exemples :

`mov al, [bx]` : transfère la donnée dont l'offset est contenu dans le registre de base BX vers le registre AL. Le segment associé par défaut au registre BX est le segment de données : on dit que l'adressage est **basé sur DS** ;

`mov al, [bp]` : le segment par défaut associé au registre de base BP est le segment de pile. Dans ce cas, l'adressage est **basé sur SS**.





□ Les instructions de transfert

- **adressage indexé** : semblable à l'adressage basé, sauf que l'offset est contenu dans un registre d'index SI ou DI, associés par défaut au segment de données.

Exemples :

`mov al, [si]` : charge le registre AL avec le contenu de la case mémoire dont l'offset est contenu dans SI ;

`mov [di], bx` : charge les cases mémoire d'offset DI et DI + 1 avec le contenu du registre BX.

Remarque : une valeur constante peut éventuellement être ajoutée aux registres de base ou d'index pour obtenir l'offset. Exemple :

`mov [si+100H], ax`

qui peut aussi s'écrire

`mov [si][100H], ax`

ou encore

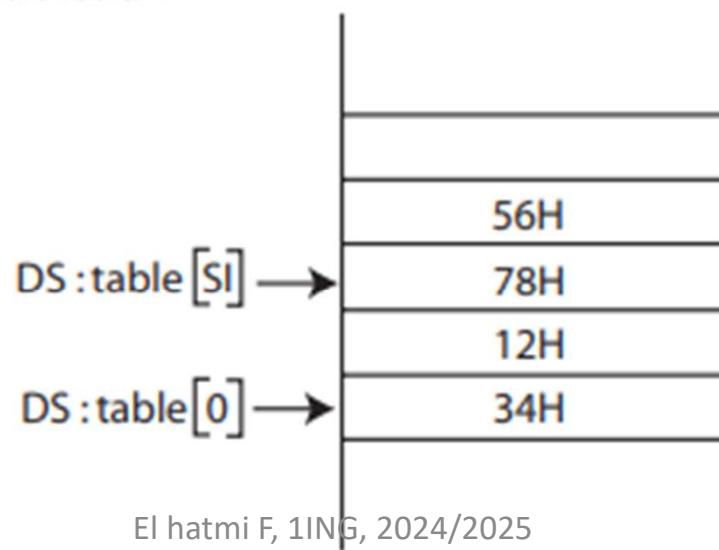
`mov 100H[si], ax`

□ Les instructions de transfert : Remarque

Les modes d'adressage basés ou indexés permettent la manipulation de tableaux rangés en mémoire. Exemple :

```
    mov si,0  
    mov word ptr table[si],1234H  
    mov si,2  
    mov word ptr table[si],5678H
```

Dans cet exemple, **table** représente l'offset du premier élément du tableau et le registre **SI** joue le rôle d'indice de tableau :





Programmation assembleur

□ Les instructions de transfert : Remarque

- **adressage basé et indexé** : l'offset est obtenu en faisant la somme d'un registre de base, d'un registre d'index et d'une valeur constante. Exemple :

```
    mov ah, [bx+si+100H]
```

Ce mode d'adressage permet l'adressage de structures de données complexes : matrices, enregistrements, ... Exemple :

```
    mov bx, 10
    mov si, 15
    mov byte ptr matrice[bx][si], 12H
```

Dans cet exemple, BX et SI jouent respectivement le rôle d'indices de ligne et de colonne dans la matrice.



Programmation assembleur

□ Les instructions arithmétiques

Les instructions arithmétiques de base sont l'**addition**, la **soustraction**, la **multiplication** et la **division** qui incluent diverses variantes. Plusieurs modes d'adressage sont possibles.

Addition : ADD opérande1,opérande2

L'opération effectuée est : opérande1 \leftarrow opérande1 + opérande2.

Exemples :

- add ah, [1100H] : ajoute le contenu de la case mémoire d'offset 1100H à l'accumulateur AH (adressage direct) ;
- add ah, [bx] : ajoute le contenu de la case mémoire pointée par BX à l'accumulateur AH (adressage basé) ;
- add byte ptr [1200H],05H : ajoute la valeur 05H au contenu de la case mémoire d'offset 1200H (adressage immédiat).

□ Les instructions arithmétiques

Soustraction : SUB opérande1,opérande2

L'opération effectuée est : opérande1 \leftarrow opérande1 – opérande2.

Multiplication : MUL opérande, où opérande est un registre ou une case mémoire.

Cette instruction effectue la multiplication du contenu de AL par un opérande sur 1 octet ou du contenu de AX par un opérande sur 2 octets. Le résultat est placé dans AX si les données à multiplier sont sur 1 octet (résultat sur 16 bits), dans (DX,AX) si elles sont sur 2 octets (résultat sur 32 bits).

Exemples :

- mov al,51
mov bl,32
mul bl
 \rightarrow AX = 51 \times 32
- mov ax,4253
mov bx,1689
mul bx
 \rightarrow (DX,AX) = 4253 \times 1689

- mov al,43
mov byte ptr [1200H],28
mul byte ptr [1200H]
 \rightarrow AX = 43 \times 28
- mov ax,1234
mov word ptr [1200H],5678
mul word ptr [1200H]
 \rightarrow (DX,AX) = 1234 \times 5678



Programmation assembleur

□ Les instructions arithmétiques

Division : DIV opérande, où opérande est un registre ou une case mémoire.

Cette instruction effectue la division du contenu de AX par un opérande sur 1 octet ou le contenu de (DX,AX) par un opérande sur 2 octets. Résultat : si l'opérande est sur 1 octet, alors AL = quotient et AH = reste ; si l'opérande est sur 2 octets, alors AX = quotient et DX = reste.

Exemples :

- `mov ax,35`
`mov bl,10`
`div bl`
→ AL = 3 (quotient) et AH = 5 (reste)

- `mov dx,0`
`mov ax,1234`
`mov bx,10`
`div bx`
→ AX = 123 (quotient) et DX = 4 (reste)

Autres instructions arithmétiques :

- ADC : addition avec retenue ;
- SBB : soustraction avec retenue ;
- INC : incrémentation d'une unité ;
- DEC : décrémentation d'une unité ;
- IMUL : multiplication signée ;
- IDIV : division signée.

□ Les instructions logiques

Ce sont des instructions qui permettent de manipuler des données au niveau des bits. Les opérations logiques de base sont :

- ET ;
- OU ;
- OU exclusif ;
- complément à 1 ;
- complément à 2 ;
- décalages et rotations.

Les différents modes d'adressage sont disponibles.

ET logique : AND opérande1,opérande2

L'opération effectuée est : opérande1 \leftarrow opérande1 ET opérande2.

Exemple :

mov al,10010110B		AL = 1 0 0 1 0 1 1 0
mov bl,11001101B	→	BL = 1 1 0 0 1 1 0 1
and al, bl		AL = <hr/> 1 0 0 0 0 1 0 0

□ Les instructions logiques

- AND : Application : masquage de bits pour mettre à zéro certains bits dans un mot.
Exemple : masquage des bits 0, 1, 6 et 7 dans un octet :

7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	1
0	0	1	1	1	1	0	0
<hr/>							
0	0	0	1	0	1	0	0

- OU : OU logique : OR opérande1,opérande2

L'opération effectuée est : opérande1 \leftarrow opérande1 OU opérande2.

Application : mise à 1 d'un ou plusieurs bits dans un mot.

Exemple : dans le mot 10110001B on veut mettre à 1 les bits 1 et 3 sans modifier les autres bits.

7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1
0	0	0	0	1	0	1	0
<hr/>							
1	0	1	1	1	0	1	1

Les instructions correspondantes peuvent s'écrire :

mov ah,10110001B

or ah,00001010B



Programmation assembleur

□ Les instructions logiques

Complément à 1 : NOT opérande

L'opération effectuée est : $\text{opérande} \leftarrow \overline{\text{opérande}}$.

Exemple :

```
mov al,10010001B          →      AL =  $\overline{10010001}$ B = 01101110B  
not al
```

Complément à 2 : NEG opérande

L'opération effectuée est : $\text{opérande} \leftarrow \overline{\text{opérande}} + 1$.

Exemple :

```
mov al,25  
mov bl,12          →      AL = 25 + (-12) = 13  
neg bl  
add al,bl
```

OU exclusif : XOR opérande1,opérande2

L'opération effectuée est : $\text{opérande1} \leftarrow \text{opérande1} \oplus \text{opérande2}$.

Exemple : mise à zéro d'un registre :

```
mov al,25          →      AL = 0  
xor al,al
```

□ Les instructions de branchement

Normalement, le processeur exécute une instruction puis passe à celle qui suit en mémoire, et ainsi de suite séquentiellement. Il arrive fréquemment que l'on veuille faire répéter au processeur une certaine suite d'instructions, comme dans le programme :

Repeter 3 fois:

 ajouter 5 au registre BX

En d'autres occasions, il est utile de déclencher une action qui dépend du résultat d'un test : Si $x < 0$:

$y = -x$

sinon

$y = x$

Les instructions de branchement (ou **saut**) permettent de modifier l'ordre d'exécution des instructions du programme en fonction de certaines conditions. Il existe 3 types de saut :

- saut inconditionnel ;
- sauts conditionnels ;
- appel de sous-programmes.



Programmation assembleur

□ Les instructions de branchement

Instruction de saut inconditionnel : JMP label

Cette instruction effectue un saut (**jump**) vers le label spécifié. Un **label** (ou **étiquette**) est une représentation symbolique d'une instruction en mémoire :

```
:}   ← instructions précédant le saut
    jmp suite
:}   ← instructions suivant le saut (jamais exécutées)
suite : ...      ← instruction exécutée après le saut
```

Exemple :

```
boucle : inc ax
          dec bx           →      boucle infinie
          jmp boucle
```

Remarque : l'instruction JMP ajoute au registre IP (pointeur d'instruction) le nombre d'octets (distance) qui sépare l'instruction de saut de sa destination. Pour un saut en arrière, la distance est négative (codée en complément à 2).

Programmation assembleur

□ Les instructions de branchement : Saut conditionnels

JZ	saute uniquement si ZF est allumé
JNZ	saute uniquement si ZF est éteint
JO	saute uniquement si OF est allumé
JNO	saute uniquement si OF est éteint
JS	saute uniquement si SF est allumé
JNS	saute uniquement si SF est éteint
JC	saute uniquement si CF est allumé
JNC	saute uniquement si CF est éteint
JP	saute uniquement si PF est allumé
JNP	saute uniquement si PF est éteint

JMP adr	EB	2	saut inconditionnel (adr. relatif).
JE adr	74	2	saut si =
JNE adr	75	2	saut si \neq
JG adr	7F	2	saut si >
JLE adr	7E	2	saut si \leq
JA adr			saut si CF = 0
JB adr			saut si CF = 1



Programmation assembleur

□ Les instructions de branchement

● Sauts conditionnels :

Les indicateurs du registre d'état (Flags) sont positionnés en fonction du dernier résultat obtenu.

Exemple :

```
:}   ← instructions précédant le saut conditionnel  
jnz suite  
:{}   ← instructions exécutées si la condition ZF = 1 est vérifiée  
suite : ...      ← instruction exécutée à la suite du saut
```

□ Fin du programme

A la fin d'un programme en assembleur, on souhaite en général que l'interpréteur de commandes du DOS reprenne le contrôle du PC. Pour cela, on utilisera la séquence de deux instructions

```
MOV AH, 4C  
INT 21
```

□ Les instructions

● Quelques instructions du 80x86 :

- Le code de l'instruction est donné en hexadécimal dans la deuxième colonne.
- La colonne suivante précise le nombre d'octets nécessaires pour coder l'instruction complète (opérande inclus).
- On note valeur une valeur sur 16 bits, et adr une adresse sur 16 bits également.

Symbol	Code Op.	Octets	
MOV AX, valeur	B8	3	AX \leftarrow valeur
MOV AX, [adr]	A1	3	AX \leftarrow contenu de l'adresse adr.
MOV [adr], AX	A3	3	range AX à l'adresse adr.
ADD AX, valeur	05	3	AX \leftarrow AX + valeur
ADD AX, [adr]	03 06	4	AX \leftarrow AX + contenu de adr.
SUB AX, valeur	2D	3	AX \leftarrow AX - valeur
SUB AX, [adr]	2B 06	4	AX \leftarrow AX - contenu de adr.
SHR AX, 1	D1 E8	2	décale AX à droite.
SHL AX, 1	D1 E0	2	décale AX à gauche.
INC AX	40	1	AX \leftarrow AX + 1
DEC AX	48	1	AX \leftarrow AX - 1
CMP AX, valeur	3D	3	compare AX et valeur.
CMP AX, [adr]	3B 06	4	compare AX et contenu de adr.
<i>Fin du programme (retour au DOS) :</i>			
MOV AH, 4C	B4 4C	2	
INT 21	CD 21	2	

□ Temps d'exécution d'une instruction

- Durée d'exécution d'un programme /

heure début – heure fin = temps passé par l'UC pour exécuter effectivement le programme (temps UC utilisateur) + temps passé pour exécuter des programmes du système d'exploitation + attente des résultats d'opérations d'E/S + temps passé pour exécuter d'autres programmes (fonctionnement "temps partagé", time-sharing).

- Périodes d'horloge par instruction (Clock Cycles Per Instruction, CPI) :

$$CPI = \frac{\text{temps UC utilisateur (en périodes d'horloge)}}{\text{nombre d'instructions du programme}}$$